# Proposed COMAL Standard

## Structured Data Statement

Throughout this document the word 'procedure' applies to both procedures and functions.

### The need for a new data statement

The present data statement that exists in the COMAL kernel has been 'lifted', almost without change, from BASIC. It is a construct that has several drawbacks, particularly in a structured language like COMAL.

The provision of the 'RESTORE label' statement allows the definition of the start of a data block, but the EOD flag connot be set until the end of all the data in a program, so it is not possible to read a block of data, in the middle of a program, using the EOD flag.

The scope of the DATA statement is at present difficult to define satisfactorially; the main problem comes with reading data within closed procedures, or whether data declared within a closed procedure is accessible to the main program.

### The proposed new format

It is proposed that a structured data-block should replace the existing DATA statement to be of the following format:

DATA statement
```
DATA <identifier>
  {<value list>}
  {<label>}
  {<value list>}
END DATA <identifier>
```

Value list
```
:<value> {,<value>}
```

Value
```
<string constant> ! <numeric constant>
```

Label
```
<identifier>:
```

To replace Restore
```
DATABLOCK  <identifier> //initialise DATA block / set data pointer
                          to start

RESTORE <identifier>    //set  DATA  pointer  to next value after
                          <identifier>
```

So an example of a block of data would be:

```
DATA my_data
```

```
  :1,2,"eek"
  :3,4,5.35
division:
  :6,7,8
  :9,"abcd",TRUE
END DATA my_data
```

In order to read a block of data from the beginning of the block the DATABLOCK <identifier> statement is used to initialise the DATA pointer (ie 'DATABLOCK my_data' in above example). Subsequent READ statements cause the data pointer to be incremented through the list of values, as with the present DATA statement. The EOD flag becomes set when the END DATA statement is met at the end of the block.

Data may be read commencing part way through a block by RESTOREing to a label that has been placed in that block (ie 'RESTORE division' in above example) after the block has been initialised.

Note that the same lables may be used within different DATA blocks without confusion since the RESTORE statement applys to the current DATA block only.

The scope of the data within a DATA block would be similar to the scope of procedures at present. That is to say that the data declared in the main program or open procedures would be globally available. Data declared within closed procedures would only be accessible within that procedure. Similarly closed procedures cannot access global data. A data block could be IMPORTed into a closed procedure, this makes available all the data, and any partitioning lables in that data block.

The global data pointer would have to be saved (on the procedure stack) when entering a CLOSED procedure to preserve it through any data reading operations that take place within the CLOSED procedure.

## Discussion

The above construct overcomes the present problems inherent with the BASIC-like DATA statements, while retaining enough similarity to make it easy to use for people used to the present arrangement; an important factor to consider when updating an existing language.

Possible extensions to the above suggestion could include allowing more than one current data pointer, to allow the reading of data from separate blocks simultaneously. However the author feels the above is sufficiently complicated for the present.